coordinamento di Andrea de Prisco

# Programmare in C su Amiga (20)

*In questa puntata completeremo lo scheletro di programma per la gestione dei menu via Intuition iniziato due mesi fa. Presenteremo inoltre nella scheda tecnica un interessante utilizzo del comando* ***list*** *di AmigaDOS*

## Introduzione

In questa puntata termina la prima fase relativa alla gestione dei menu in C. Completeremo cioè lo scheletro iniziato nella scorsa puntata, aggiungendovi un esempio di struttura a menu di tipo classico. Si tratta di una struttura a due livelli (menu e voci), in cui i vari elementi sono rappresentati da testi o titoli che dir si voglia, un po' come quella che si può ottenere con l'istruzione **MENU** dell'*AmigaBasic*. Questo scheletro può quindi già essere utilizzato per una classe abbastanza vasta di programmi. In seguito aggiungeremo anche strutture più sofisticate, in modo tuttavia da garantire quella filosofia di sviluppo modulare che abbiamo introdotto fin dall'inizio, e che ci permette di togliere ed aggiungere funzioni con un impatto trascurabile sul resto del codice.

Prima di entrare nel vivo dell'argomento di questo mese, tuttavia, vediamo di rispondere al quesito posto nella scorsa puntata e relativo all'utilizzo della **grep.lib** in un programma in C (vedi nota 1).

Il codice in figura 5 (della 19ª puntata) è corretto, in linea di massima, ma non è il più adatto al tipo di ricerca che si sta effettuando. Dato che stiamo cercando una stringa di caratteri in un testo, è meglio effettuare la scansione linea per linea, e non a blocchi di un certo numero di byte.

Quello che ci interessa è infatti in quale linee del testo si trova la stringa che cerchiamo, mentre se si utilizza il codice mostrato in figura, *GREP* ci riporta in quale blocco di *tot* byte è stata trovata quella determinata stringa, il che non è particolarmente utile dal nostro punto di vista. Il programma va allora modificato aprendo il file in modo testo (non binario), ed utilizzando la **fgets()** invece della **fread()**, e specificando un buffer più lungo della più lunga linea di testo che ci si aspetta di trovare. Va poi gestita la possibilità che una stringa sia a cavallo di due linee. Questo si fa agganciando nel modo opportuno una linea alla precedente (sostituendo un **newline** **\n** con uno spazio), ed effettuando la ricerca sulla stringa risultante. Così ogni linea viene scandita due volte, ma almeno siamo sicuri di trovare *tutte* le occorrenze che ci interessano.

Ed ora vediamo come si identifica un menu in Intuition.

## Identificativi vari

Quando Intuition ci notifica un evento di tipo **MENUPICK**, cioè della selezione da parte dell'utente di una voce o di una sottovoce, il codice del Messaggio (Campo **Code**) contiene un valore che ci permette di scoprire quale menu, voce e/o sottovoce sia stata selezionata (o meglio la prima selezione, in caso di selezione multiple).

Se tale codice è diverso da **MENUNULL**, allora è possibile usare tre macro di sistema per estrarre da tale valore gli identificativi del menu, della voce e della sottovoce, se esiste. Le tre macro sono riportate in figura 1 e sono:

**MENUNUM()** estrae l'identificativo del menu
**ITEMNUM()** estrae l'identificato della voce
**SUBNUM()** estrae l'identificativo della sottovoce.

Tali identificativi corrispondono a quelli definiti dall'utente nel programma che costruisce i menu. La figura summenzionata mostra anche alcune costanti più o meno utili che possono essere utilizzate per ricostruire un codice a partire dai singoli identificativi, grazie ad altre tre macro che operano inversamente alle precedenti, o per effettuare dei test sugli identificativi stessi (come **NOSUB**).

Attualmente il codice dei menu è una parola da 16 bit utilizzata per contenere tutti e tre i possibili identificativi, nel modo rappresentato in tabella A.

Tuttavia è bene non fare assunzioni sul formato di tale codice; dato che nulla vieta gli sviluppatori del sistema operativo di modificarlo in una versione futura. Se si vuole essere sicuri di scrivere un codice sempre allineato al sistema, indipendentemente dalle versioni, è bene utilizzare sempre le interfacce *ufficiali*. Nel nostro caso questo vuol dire utilizzare le macro di sistema or ora presentate.

## Il programma scheletro

Veniamo adesso al programma scheletro. Dall'ultima volta che lo abbiamo visto è cresciuto parecchio. Facendo riferimento alla figura 2 vediamo quali sono le principali modifiche. Prendiamo in esame la parte comune, prima di tutto, quella cioè che contiene le dichiarative globali dei tipi, delle costanti e delle variabili (semplici e strutture).

Innanzi tutto, per comodità, abbiamo aggiunto alcuni nuovi tipi. Anche la lista dei prototipi delle funzioni interne è cresciuta, ma di queste parleremo più in dettaglio tra un po'. Nel blocco relativo ai *puntatori alle principali strutture* abbiamo aggiunto un nuovo puntatore, molto importante, usato per allocare dinamicamente tutta una serie di strutture in modo molto flessibile, come vedremo parlando della **StartAll()**. Abbiamo quindi aggiunto due nuove mascherine e finalmente riempito il blocco più importante: quello relativo alla definizione dei menu.

E qui è bene spendere due parole al riguardo .

Benché le regole con cui si devono costruire i menu e riempire le strutture da passare ad Intuition siano ben definite, non esiste alcuna tecnica «ufficiale» da utilizzare per la definizione di menu. Ne risulta che ognuno fa a modo suo e, dato che è necessario riempire molti campi e legare fra loro le strutture in una struttura gerarchica che può risultare anche abbastanza complessa, succede spesso che il codice risultante sia abbastanza contorto, illeggibile e poco flessibile.

Partendo da una tecnica sviluppata da *John T. Draper* e da me sviluppata e migliorata in modo da incrementarne la flessibilità (vedi nota 5) e da ridurre le dimensioni del codice risultante, ho definito una tecnica di definizione dei menu che è appunto quella mostrata nello scheletro in questione.

Ne consegue che le dichiarative utilizzate nel codice presentato, fanno principalmente parte della tecnica in questione, piuttosto che delle regole di sviluppo di Intuition. Ovviamente esiste anche un *contro*. La tecnica qui presentata è particolarmente utile quando si deve costruire una struttura a menu del tipo di quella creata dal comando **MENU** dell'AmigaBasic, anche se più complessa e con la possibilità di gestire molte

```
/* ------------------------------------------------------------------ *
 * --- Macro per analizzare l'identificativo di un menù           --- *
 * ------------------------------------------------------------------ */
#define MENUNUM(n)      (n & 0x1F)
#define ITEMNUM(n)      ((n >> 5) & 0x003F)
#define SUBNUM(n)       ((n >> 11) & 0x001F)

/* ------------------------------------------------------------------ *
 * --- Macro per costruire l'identificativo di un menù            --- *
 * ------------------------------------------------------------------ */
#define SHIFTMENU(n)    (n & 0x1F)
#define SHIFTITEM(n)    ((n & 0x3F) << 5)
#define SHIFTSUB(n)     ((n & 0x1F) << 11)

/* ------------------------------------------------------------------ *
 * --- Costanti usate nell'analisi dell'identificativo di un menù --- *
 * ------------------------------------------------------------------ */
#define NOMENU      0x001F
#define NOITEM      0x003F
#define NOSUB       0x001F
#define MENUNULL    0xFFFF

/* ------------------------------------------------------------------ *
 * --- Costanti usate nel posizionamento del simbolo di selezione --- *
 * ------------------------------------------------------------------ */
#define CHECKWIDTH      19
#define LOWCHECKWIDTH   13

/* ------------------------------------------------------------------ *
 * --- Costanti usate nel posizionamento del comando scorciatoia (A) --- *
 * ------------------------------------------------------------------ */
#define COMMWIDTH       27
#define LOWCOMMWIDTH    16
```

*Figura 1*
*Macro e costanti relative ad i menu.*

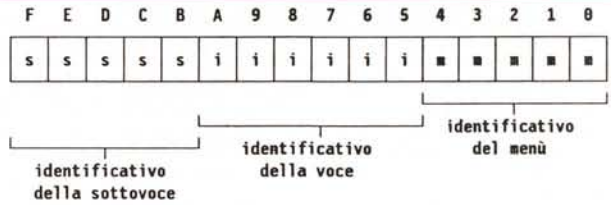


*Tabella A*

più opzioni; tuttavia essa va ulteriormente modificata nel momento in cui sorga l'esigenza di utilizzare nei menu immagini o voci miste (testo più immagini). Di fatto rimane sempre un'ottima base di partenza anche per strutture più sofisticate.

Vediamo quindi cosa andiamo a definire nel blocco principale.

Innanzi tutto definiamo tre costanti: **HITEM**, che rappresenta l'altezza di un elemento *voce*, **CHARWIDTH** che rappresenta un fattore moltiplicativo per dimensionare l'elemento *menu* in funzione della lunghezza in caratteri del testo usato per rappresentarlo, ed infine **MENUFLAGS** che contiene i segnalatori IDCMP che attiveranno gli eventi a cui siamo interessati.

La struttura seguente è una base da usare per tutte le strutture **IntuiText** utilizzate nella definizione delle voci.

Quindi definiamo un set di costanti, che rappresentano:

- il numero dei menu
- gli identificativi dei menu [*menu number*]
- il numero delle voci
- gli identificativi delle voci [*item number*].

Queste costanti verranno usate dalla procedura che analizza l'evento di tipo **MENUPICK**.

Successivamente definiamo il puntatore alle strutture menu, e quelli relativi ad ogni singola lista di voci con la rispettiva lista di testi. Questo perché, dato che tra menu e voci il numero di strutture da definire può essere alquanto elevato, ho ritenuto più conveniente allocare tali strutture dinamicamente, piuttosto che staticamente, riducendo così le dimensioni del compilato.

Le costanti successive contengono i testi per i singoli menu, e la larghezza delle voci con gli attributi di evidenziazione. Di quest'ultimi parleremo nella prossima puntata.

Le strutture seguenti servono a definire i testi per le singole voci. L'ultima altro non è che un vettore di tutte i testi definiti.

E vediamo adesso le singole procedure. Il **main()**, come si può vedere in figura, non è cambiato.

Diverso è il discorso per la maggior parte delle altre *routines*.

## *StartAll()*

Qui abbiamo aggiunto una serie di allocazioni di memoria delle varie strutture **Menu, MenuItem** e **INTUITEXT** che useremo per definire la struttura gerarchica a menu. Tuttavia, invece di usare la classica **AllocMem()**, abbiamo preferito la più flessibile **AllocRemember()** fornitaci da Intuition. Questa funzione alloca memoria come la sua cugina di EXEC, ma in aggiunta mantiene traccia di tutti i blocchi che ha allocato e che sono stati associati ad uno specifico puntatore di riferimento [*anchor pointer*] definito dal programmatore. Il vantaggio è che indipendentemente dal numero di aree di memoria allocate e dalle loro caratteristi-

## *NOTE*

**1.** Il titolo della figura 5 della scorsa puntata non è *Due esempi di..., ma Esempio di...*

**2.** In realtà si poteva anche usare **ModifyIDCMP(w,0L);** se non addirittura non chiamare del tutto la funzione, dato che siamo comunque in chiusura, ma ho preferito lo stesso mostrare la tecnica formalmente più corretta da usare nel caso si intenda continuare il programma dopo la chiusura dei menu.

**3.** Nel manuale *ENHANCED SOFTWARE featuring AmigaDOS Version 1.3* venduto con i dischetti della versione 1.3 del sistema operativo, si afferma che nel caso di tre coppie **%S** la seconda viene sostituita dal nome del file, mentre le altre due sono sostituite dal cammino.
Questo non è corretto, come è facile verificare lanciando il comando
**1> list l format="1 [%S], 2 [%S], 3 [%S]."**
D'altro canto è più pratico com'è in realtà, dato che se si lancia un comando diadico, cioè a due parametri di ingresso, il primo generalmente contiene il nome completo del file (nome più cammino), il secondo ha più probabilità di utilizzare di nuovo il nome del file piuttosto che il cammino per raggiungerlo.

**4.** È sempre opportuno definirsi *ad hoc* una convenzione per i nomi dei file temporanei *[naming convention]*. Questo principalmente per due motivi:
- il primo è che così uno è in grado di identificare immediatamente se un file è temporaneo o meno, ed eventualmente cancellarlo se, per un qualche motivo non è stato cancellato dal processo che lo aveva creato;
- il secondo è che generalmente i file temporanei sono creati da procedure script *AmigaDOS* o *ARexx*, ed in un sistema multitasking è bene assicurarsi con una convenzione che due procedure non utilizzino lo stesso nome per due scopi differenti.

La convenzione da me adottata (ma ognuno può scegliersi la sua) è la seguente: *un file temporaneo ha il nome sempre incluso tra parentesi quadre, e come prefisso i primi tre caratteri del nome del processo che lo ha creato seguiti da un punto, nel caso ci si voglia garantire al massimo l'univocità:*
**[temp] [ren.in] [ren.out] ...e così via.**

**5.** Per *flessibilità* di un programma si intende la capacità dello stesso di venir modificato in termini di *funzionalità* riducendo al minimo l'impatto sul codice preesistente e rendendo facile e chiara l'individuazione dei blocchi da modificare aumentando così la mantenibilità del programma e riducendo i tempi di sviluppo.

```
/*************************************************************************
**  Programmare in C su Amiga  (c) 1989  Dario de Judicibus - Roma (I)   **
** --------------------------------------------------------------------- **
**  Scheletro di un programma di gestione dei menù.                      **
**                                                                       **
**    Questo programma crea una struttura a menù da associare ad una     **
**    finestra che poi apre sullo schermo del WorkBench. Il programma    **
**    è altamente strutturato in modo da presentare uno scheletro        **
**    flessibile da dettagliare in più fasi successive.                  **
**                                                                       **
**  Riconoscimento: questo scheletro è basato in parte su di una tecnica **
**  sviluppata da John T. Draper - Sausalito (USA). Molte procedure sono **
**  il risultato di studi compiuti dal sottoscritto e da altri sviluppa- **
**  tori europei ed americani, solo l'idea base di John è stata mantenu- **
**  ta e di fatto ampiamente modificata. Il codice risultante è così     **
**  molto più flessibile di quello di John, oltre al fatto di generare   **
**  un modulo eseguibile molto più ridotto in dimensioni.                **
**                                                                       **
*************************************************************************/

 #include "exec/types.h"
 #include "exec/memory.h"
 #include "intuition/intuition.h"
 #include "graphics/gfxmacros.h"
 #include "proto/exec.h"
 #include "proto/intuition.h"
 #include "proto/graphics.h"
 #include "stdio.h"
 #include "stdlib.h"
 #include "string.h"

 /*
 **  Tipi
 */
 typedef struct Node         NODE;
 typedef struct Message      EMSG;
 typedef struct IntuiMessage IMSG;
 typedef struct IntuiText    ITXT;
 typedef struct Menu         MENU;
 typedef struct MenuItem     ITEM;
```

```
/*
**  Prototipi delle funzioni interne al programma
*/
 void StartAll     ( void );
 void CloseAll     ( void );
 void BuildMenus   ( void );
 void SetupMenu    ( MENU *, MENU *, BYTE *, ITEM * );
 void SetupItemList( ITEM *, int, SHORT, USHORT, ITXT *, UBYTE **, ITEM * );
 void LetsGo       ( void );
 int  HandleEvent  ( IMSG * );
 void CloseSafelyWindow ( struct Window *, struct TextFont * );
/* ------------------------------------------------------------ STUBs --- */
 int  H_MenuVerify   ( IMSG * );
 int  H_MenuPick     ( IMSG * );
 int  H_MouseButtons ( IMSG * );
/* ---------------------------------------------------------------------- */
 int  H_CloseWindow     ( IMSG * );

/*
**  Costanti
*/
 #define IREV 0
 #define GREV 0
 #define INAME "intuition.library"
 #define GNAME "graphics.library"
 #define DJ_COLS 400
 #define DJ_ROWS 150
 #define DJ_TITL "Esempio di gestione dei menù [DdJ]"
 #define GOAHEAD 1
 #define CLOSEME 0

/*
**  Caratteristiche della finestra: gadget di CHIUSURA, di PROFONDITA',
**  di SPOSTAMENTO, restauro automatico intelligente, tipo GZZ, attiva.
*/
 #define DJ_BASE  WINDOWCLOSE|WINDOWDEPTH|WINDOWDRAG
 #define DJ_SPEC  GIMMEZEROZERO|SMART_REFRESH|ACTIVATE

/*
**  Puntatori alle principali strutture
*/
 struct IntuitionBase *IntuitionBase;
 struct GfxBase *GfxBase;
 struct Window *w;
 struct RastPort *rp;
 struct MsgPort *up;
 struct Remember *rememory;
 IMSG *imsg;
```

```
/*
**  Strutture di definizione della finestra e dei menù
*/
 struct NewWindow nw =
 {
   20, 20, DJ_COLS, DJ_ROWS,  /* posizione e dimensioni della finestra  */
   0, 1,                      /* colore delle penne di fondo e di segno */
   CLOSEWINDOW,               /* Segnalatori IDCMP: gadget di chiusura  */
   DJ_BASE|DJ_SPEC,           /* caratteristiche della finestra         */
   NULL, NULL, DJ_TITL,       /* gadget, checkmark, titolo              */
   NULL, NULL, 0, 0, 0, 0,    /* schermo, superbitmap, dimensioni min.  */
   WBENCHSCREEN               /* da aprire sullo schermo del WorkBench  */
 };
 ULONG SavFlags;              /* Per salvare i segnalatori IDCMP        */

/*
**  Maschere di controllo
*/
 #define MSK_INT 0x0001
 #define MSK_GFX 0x0002
 #define MSK_WIN 0x0004
 #define MSK_MEM 0x0008
 #define MSK_MST 0x0010
 UWORD  mask = 0x0000;

/* ===================================================================== **
**                STRUTTURE DI DEFINIZIONE PER I MENU'                   **
** ===================================================================== */

 #define HITEM 10
 #define CHARWIDTH 10
 #define MENUFLAGS (MENUPICK|MOUSEBUTTONS)

 ITXT basetext =       /* Questa struttura fa da base ai testi delle voci */
 {
   0, 1,               /* Penne per il tratto e per lo sfondo             */
   JAM2, 5,            /* Modo grafico e spostamento dal bordo sinistro   */
   0,                  /* Spostamento dal bordo superiore                 */
   NULL,               /* Font usato, se diverso da quello di sistema.    */
   NULL,               /* Testo (da riempire)                             */
   NULL                /* Eventuale struttura successiva                  */
 };


/*
**  Numero dei menù e loro identificativi, con le rispettive voci
**  Ovviamente le costanti possono avere anche nomi "parlanti", come
**  EDIT_MENU, EDI_COPY, EDI_PASTE, e così via...
*/
 #define MENU_NUM  3
 #define MENU_100  0
 #define MENU_200  1
 #define MENU_300  2

 #define ITEM_1NM  5
 #define ITEM_110  0
 #define ITEM_120  1
 #define ITEM_130  2
 #define ITEM_140  3
 #define ITEM_150  4

 #define ITEM_2NM  3
 #define ITEM_210  0
 #define ITEM_220  1
 #define ITEM_230  2

 #define ITEM_3NM  4
 #define ITEM_310  0
 #define ITEM_320  1
 #define ITEM_330  2
 #define ITEM_340  3

/*
**  Dato che le strutture per i menù e le voci sono tante, definiamo
**  solo i puntatori, ed allochiamole dinamicamente in seguito.
*/
 MENU *menulist;               /* Puntatore al vettore dei menù         */
 ITEM *itemlist[MENU_NUM];     /* Vettore dei puntatori ai vettori voci */
 ITXT *itemtext[MENU_NUM];     /* Vettore di tutte le strutture testi   */

/*
**  Menù: testi
*/
 #define MENU_1TX  "Primo menù"
 #define MENU_2TX  "Secondo menù"
 #define MENU_3TX  "Terzo menù"

/*
**  Voci: larghezze, caratteristiche e testi
*/
 #define ITEM_1WD  148
 #define ITEM_2WD  148
 #define ITEM_3WD  148
 #define ITEM_1FL  (ITEMTEXT|ITEMENABLED|HIGHBOX)
 #define ITEM_2FL  (ITEMTEXT|ITEMENABLED|HIGHCOMP)
 #define ITEM_3FL  (ITEMTEXT|ITEMENABLED|HIGHBOX|CHECKIT)
```

```
UBYTE *item_tit1[] =
{
   "M1: prima voce"
  ,"M1: seconda voce"
  ,"M1: terza voce"
  ,"M1: quarta voce"
  ,"M1: quinta voce"
};
UBYTE *item_tit2[] =
{
   "M2: prima voce"
  ,"M2: seconda voce"
  ,"M2: terza voce"
};
UBYTE *item_tit3[] =
{
   "M3: prima voce"
  ,"M3: seconda voce"
  ,"M3: terza voce"
  ,"M3: quarta voce"
};
UBYTE **itemname[] =
{
  item_tit1, item_tit2, item_tit3
};

/*************************************************************************
*************************************************************************
** main: programma principale                                          **
*************************************************************************
*************************************************************************/
void main()
{
  StartAll   ();    /* Effettuiamo le chiamete di partenza.            */
  BuildMenus ();    /* Costruiamo i menù da associare alla finestra.   */
  LetsGo     ();    /* Va bene. E' tutto pronto. Andiamo!              */
  CloseAll   ();    /* Finito. Chiudiamo tutto.                        */
}

/*************************************************************************
** StartAll: chiamate di partenza                                      **
*************************************************************************/
void StartAll()
{
 /*
 **  Apre le librerie (Intuition & Graphics) e la finestra
 */
  IntuitionBase = (struct IntuitionBase *)OpenLibrary(INAME,IREV);
  if (IntuitionBase == NULL) CloseAll();
  mask |= MSK_INT;
  GfxBase = (struct GfxBase *)OpenLibrary(GNAME,GREV);
  if (GfxBase == NULL) CloseAll();
  mask |= MSK_GFX;
  w = (struct Window *)OpenWindow(&nw);
  if (w == NULL) CloseAll();
  mask |= MSK_WIN;
  rp = w->RPort;      /* RastPort per la grafica */
  up = w->UserPort;   /* Porta utente per IDCMP  */

 /*
 **  Alloca dinamicamente le strutture per i menù e le voci
 **  Usa AllocRemember() per una gestione ordinata e flessibile
 */
  rememory = NULL;

 /*
 **  Menù
 */
  menulist = (MENU *)AllocRemember(&rememory,
             MENU_NUM * sizeof(MENU), MEMF_CLEAR);
  if (menulist == NULL) CloseAll();
  mask |= MSK_MEM;                  /* Nota, basta la prima allocazione */

 /*
 **  Voci
 */
  itemlist[0] = (ITEM *)AllocRemember(&rememory,
                ITEM_1NM * sizeof(ITEM), MEMF_CLEAR);
  if (itemlist[0] == NULL) CloseAll();
  itemlist[1] = (ITEM *)AllocRemember(&rememory,
                ITEM_2NM * sizeof(ITEM), MEMF_CLEAR);
  if (itemlist[1] == NULL) CloseAll();
  itemlist[2] = (ITEM *)AllocRemember(&rememory,
                ITEM_3NM * sizeof(ITEM), MEMF_CLEAR);
  if (itemlist[2] == NULL) CloseAll();

 /*
 **  Testi
 */
  itemtext[0] = (ITXT *)AllocRemember(&rememory,
                ITEM_1NM * sizeof(ITXT), MEMF_CLEAR);
  if (itemtext[0] == NULL) CloseAll();
  itemtext[1] = (ITXT *)AllocRemember(&rememory,
                ITEM_2NM * sizeof(ITXT), MEMF_CLEAR);
  if (itemtext[1] == NULL) CloseAll();
  itemtext[2] = (ITXT *)AllocRemember(&rememory,
                ITEM_3NM * sizeof(ITXT), MEMF_CLEAR);
  if (itemtext[2] == NULL) CloseAll();
}

/*************************************************************************
** CloseAll: chiamate di chiusura                                      **
*************************************************************************/
void CloseAll()  /* ordine inverso rispetto StartAll()!!! */
{
  if (mask & MSK_MST)
  {
    ClearMenuStrip(w);
    ModifyIDCMP(w,SaveFlags);
  }
  if (mask & MSK_MEM) FreeRemember(&rememory,TRUE);
  if (mask & MSK_WIN) CloseWindow(w);
  if (mask & MSK_GFX) CloseLibrary(GfxBase);
  if (mask & MSK_INT) CloseLibrary(IntuitionBase);
  Exit(0);
}

/*************************************************************************
** CloseSafelyWindow: chiude una finestra che condivide con altre la   **
**                    stessa porta utente nel modo più sicuro          **
**                    Per l'ultima usa invece CloseWindow().           **
*************************************************************************/
void CloseSafelyWindow(wptr,fptr)
  struct Window *wptr;
  struct TextFont *fptr;
{
  IMSG *scan;

  /*
  **  Si assume di aver già cancellato la barra menù relativa alla finestra
  */
  if (wptr == NULL) return; /* Logica della scatola nera: per sicurezza  */

  /*
  **  Dato che stiamo lavorando su una lista di sistema è necessario
  **  disabilitare temporaneamente il multitasking.
  */
  Forbid();               /* Blocca il multitasking.                     */

  /*
  **  Ciclo sulla lista dei messaggi arrivati alla porta utente. Quelli
  **  relativi alla finestra da chiudere sono rimossi.
  */
  for (scan = (IMSG *)up->mp_MsgList.lh_Head;      /* Inizio lista       */
       (IMSG *)scan->ExecMessage.mn_Node.ln_Succ;  /* Successivo è NULLO */
       scan = (IMSG *)scan->ExecMessage.mn_Node.ln_Succ)  /* Successivo  */
  {
    if (scan->IDCMPWindow == wptr) /* Il messaggio è per questa finestra */
    {
     /*
     **  Usa Remove() invece di GetMsg() dato che quest'ultima toglie
     **  SEMPRE il messaggio in testa alla lista.
     */
      Remove((NODE *)scan);   /* OK. Cancelliamolo dalla coda messaggi.. */
      ReplyMsg((EMSG *)scan); /* e rispondiamo al mittente.              */
    }
  }

  wptr->UserPort = NULL;  /* Evita che Intuition deallochi la porta ut.  */
  ModifyIDCMP(wptr,NULL); /* OK. Adesso effettuiamo la chiusura "logica" */

  Permit();               /* Ripristina il multitasking.                 */

 /*
 **  OK. Ora possiamo chiudere tranquillamente la finestra
 */
  CloseWindow(wptr);
  if (fptr) CloseFont(fptr); /* Nel caso avvessimo caricato un font      */
}
```

*(continua a pag. 204)*

*Figura 2 - Il programma scheletro.*

(segue da pag. 203)

```
/*****************************************************************************
** LetsGo: OK. Blocco principale di controllo                               **
*****************************************************************************/
void LetsGo(void)
{
 /*
 **  Svuotiamo la coda messaggi o mettiamoci in attesa del successivo
 */
  FOREVER  /* Ciclo infinito: si interrompe con "break" */
  {
    if ((imsg = (IMSG *)GetMsg(up)) == NULL)  WaitPort(up);
    else if (HandleEvent(imsg) == CLOSEME)    break;
  }
}

/*****************************************************************************
** BuildMenus: Costruisce i menù                                            **
*****************************************************************************/
void BuildMenus()
{
 /*
 **  Definiamo i menù
 */
  SetupMenu(&menulist[0],NULL        ,MENU_1TX,itemlist[0]);
  SetupMenu(&menulist[1],&menulist[0],MENU_2TX,itemlist[1]);
  SetupMenu(&menulist[2],&menulist[1],MENU_3TX,itemlist[2]);

 /*
 **  Definiamo le voci
 */
  SetupItemList(itemlist[0],ITEM_1NM,ITEM_1WD,ITEM_1FL,
                itemtext[0],itemname[0],NULL);
  SetupItemList(itemlist[1],ITEM_2NM,ITEM_2WD,ITEM_2FL,
                itemtext[1],itemname[1],NULL);
  SetupItemList(itemlist[2],ITEM_3NM,ITEM_3WD,ITEM_3FL,
                itemtext[2],itemname[2],NULL);

 /*
 **  Fatto! E adesso associamo il tutto alla finestra.
 */
  SetMenuStrip(w,&menulist[0]);
  SaveFlags = w->IDCMPFlags;
  ModifyIDCMP(w,SaveFlags|MENUFLAGS);
  mask |= MSK_MST;
}
```

```
/*****************************************************************************
** SetupMenu: Costruisce un menù                                            **
*****************************************************************************/
void SetupMenu(newmenu,prevmenu,menuname,ilist)
  MENU   *newmenu,          /* Puntatore al menù da definire            */
         *prevmenu;         /* Puntatore ad un eventuale menù precedente */
  BYTE   *menuname;         /* Titolo del menù                          */
  ITEM   *ilist;            /* Puntatore alla lista delle voci          */
{
  newmenu->NextMenu  = NULL;        /* Per ora è l'ultimo...            */
  newmenu->LeftEdge  = 0;           /* Ed anche il primo...             */
  newmenu->TopEdge   = 0;           /* Sempre a zero                    */
  newmenu->Height    = 10;          /* Altezza della barra in pixel     */
  newmenu->Flags     = MENUENABLED; /* Il menù è abilitato              */
  newmenu->MenuName  = menuname;    /* Titolo del menù: solo testo      */
  newmenu->FirstItem = ilist;       /* Puntatore alla lista delle voci  */

  if (prevmenu)                     /* Ce n'è uno prima? Se sì...       */
  {
    prevmenu->NextMenu = newmenu; /* Lega il nuovo menù al precedente    */
    newmenu->LeftEdge = prevmenu->LeftEdge + prevmenu->Width;
  }

  newmenu->Width     = strlen(menuname) * CHARWIDTH;
}
```

```
/*****************************************************************************
** SetupItemList: costruisce una lista di voci                              **
*****************************************************************************/
void SetupItemList(ilist,itemnum,itemwidth,itemflags,it,itemname,slist)
  ITEM    *ilist;           /* Lista delle voci: vettore                */
  int     itemnum;          /* Numero di voci nella lista               */
  SHORT   itemwidth;        /* Larghezza dell'elemento                  */
  USHORT  itemflags;        /* Caratteristiche dell'elemento            */
  ITXT    *it;              /* Vettore di strutture IntuiText da usare  */
  UBYTE   *itemname[];      /* Titoli delle voci della lista            */
  ITEM    *slist;           /* Puntatore alla lista delle sottovoci     */
{
  int i;

  for (i = 0; i < itemnum; i++)
  {
    ilist[i].NextItem    = &ilist[i+1];  /* Lega alla voce successiva    */
    ilist[i].LeftEdge    = 0;            /* Sempre zero                  */
    ilist[i].TopEdge     = HITEM*i;      /* Distanza dal bordo superiore */
    ilist[i].Height      = HITEM-2;      /* Altezza dell'elemento        */
    ilist[i].Flags       = itemflags;    /* Caratteristiche della voce   */
    ilist[i].MutualExclude = 0x0000;     /* Tutti indipendenti, per ora  */
    ilist[i].Command     = 0;            /* Nessun comando, per ora      */
    ilist[i].SubItem     = slist;        /* Puntatore alle sottovoci     */
    ilist[i].NextSelect  = MENUNULL;     /* Per le selezioni multiple    */

    ilist[i].Width       = itemwidth +   /* Larghezza dell'elemento      */
       ((itemflags & CHECKIT)  ? CHECKWIDTH : 0) +  /* Marcatore?       */
       ((ilist[i].Command != 0) ? COMMWIDTH : 0) ;  /* Scorciatoia?     */

    /*
    **  Cloniamo la struttura base ed assegnamo al campo "IText" il titolo
    **  della voce. Se è previsto un "checkmark", lasciamo sufficiente spazio
    **  a destra.
    */
    it[i]                = basetext;
    it[i].IText          = itemname[i];
    it[i].LeftEdge       = it[i].LeftEdge +
       ((itemflags & CHECKIT)  ? CHECKWIDTH : 0) ;  /* Marcatore?       */
    ilist[i].ItemFill    = (APTR)&it[i];

    ilist[i].SelectFill  = NULL;         /* Testo alternativo: nullo     */
  }
  ilist[itemnum-1].NextItem = NULL;     /* Ultimo elemento              */
}
```

```
/*****************************************************************************
** HandleEvent: gestione dei messaggi da Intuition                          **
*****************************************************************************/
int HandleEvent(msg)
  IMSG *msg;
{
  IMSG localmsg;     /* Questa è una fotocopia del messaggio ricevuto */
  int  result;       /* Questo è il valore da restituire al chiamante */
                     /* Fotocopiamo il messaggio originale            */
  CopyMem((char *)msg,(char *)&localmsg,sizeof(IMSG));

  /* ------------------------------------------------------------------ *
   * ATTENZIONE: i controlli di tipo VERIFY vanno messi PRIMA di        *
   *             rispondere al messaggio, altrimenti non servono a      *
   *             niente! Potevamo anche usare msg->Class, ovviamente.   *
   * ------------------------------------------------------------------ */
  switch (localmsg.Class)
  {
    case MENUVERIFY  : result = H_MenuVerify   (&localmsg);   break;
  }

  ReplyMsg((struct Message *)msg); /* OK. Adesso possiamo rispondere. */

  /* ------------------------------------------------------------------ *
   * ATTENZIONE: da questo punto in poi il messaggio puntato da "msg"   *
   *             non è più disponibile a questo task.                   *
   *             Useremo la copia locale salvata in "localmsg".         *
   *             MENUVERIFY è ripetuto per evitare che l'assegnazione   *
   *             di "result" in "default:" si sovrapponga a quella      *
   *             precedente. Questo è uno dei tanto modi per evitarlo.  *
   * ------------------------------------------------------------------ */

  switch (localmsg.Class)
  {
    case CLOSEWINDOW : result = H_CloseWindow  (&localmsg);   break;
    case MENUPICK    : result = H_MenuPick     (&localmsg);   break;
    case MOUSEBUTTONS: result = H_MouseButtons (&localmsg);   break;
    case MENUVERIFY  : /* # già trattato in precedenza # */   break;
    default          : result = GOAHEAD                ;     break;
  };
  return (result);
}

/*****************************************************************************
** STUB ROUTINES: per il momento non fanno niente                           **
*****************************************************************************/
int H_MenuVerify    (msg)  IMSG *msg; { return(GOAHEAD); }
int H_MouseButtons  (msg)  IMSG *msg; { return(GOAHEAD); }
```

```
/*****************************************************************************
** H_MenuPick: gestisce l'evento MENUPICK                                   **
*****************************************************************************/
 int H_MenuPick(msg)
   IMSG *msg;
 {
   MENU *strip;
   ITEM *select;
   struct Window *whichwin;
   USHORT id, menunum, itemnum, subnum;

   /*
   **  E' stata fatta effettivamente una selezione? Se sì, allora
   **  determiniamo a quale finestra si riferisce, e quindi a quale
   **  menù, così possiamo usare questa procedura per gestire più menù
   **  appartenenti a più finestre.
   */
    id = msg->Code;                    /* Identificativo selezione        */
    while (id != MENUNULL)             /* id = sssssiiiiiimmmmm  (16 bits) */
    {
      menunum = MENUNUM(id);           /* Identificativo del menù         */
      itemnum = ITEMNUM(id);           /* Identificativo della voce       */
      subnum  = SUBNUM(id);            /* Identificativo della sottovoce  */

      whichwin = msg->IDCMPWindow;     /* Indirizzo della finestra        */
      strip = whichwin->MenuStrip;     /* Indirizzo della barra dei menù  */
      select = ItemAddress(strip,id);  /* Indirizzo della voce selezionata */

      /*
      **  OK. Vediamo cosa è stato selezionato. Il caso riportato è relativo
      **  ad una sola finestra, ma può essere facilmente generalizzato a più
      **  finestre nel modo seguente:
      **
      **  if      (whichwin == win_001)  * Codici menù della finestra 001   *
      **  else if (whichwin == win_002)  * Codici menù della finestra 002   *
      **  ...
      **  else if (whichwin == win_00N)  * Codici menù della finestra 00N   *
      **  else                           * Errore: finestra sconosciuta!    *
      */




      /*
      **  BLOCCO PER LA GESTIONE DEI CODICI
      */
       switch (menunum)
       {
         case  MENU_100:
               printf("Menù [%s]\n",MENU_1TX);
               switch (itemnum)
               {
                 case  ITEM_110: printf("Voce [%s]\n",itemname[0][0]); break;
                 case  ITEM_120: printf("Voce [%s]\n",itemname[0][1]); break;
                 case  ITEM_130: printf("Voce [%s]\n",itemname[0][2]); break;
                 case  ITEM_140: printf("Voce [%s]\n",itemname[0][3]); break;
                 case  ITEM_150: printf("Voce [%s]\n",itemname[0][4]); break;
               }
               break;
         case  MENU_200:
               printf("Menù [%s]\n",MENU_2TX);
               switch (itemnum)
               {
                 case  ITEM_210: printf("Voce [%s]\n",itemname[1][0]); break;
                 case  ITEM_220: printf("Voce [%s]\n",itemname[1][1]); break;
                 case  ITEM_230: printf("Voce [%s]\n",itemname[1][2]); break;
               }
               break;
         case  MENU_300:
               printf("Menù [%s]\n",MENU_3TX);
               switch (itemnum)
               {
                 case  ITEM_310: printf("Voce [%s]\n",itemname[2][0]); break;
                 case  ITEM_320: printf("Voce [%s]\n",itemname[2][1]); break;
                 case  ITEM_330: printf("Voce [%s]\n",itemname[2][2]); break;
                 case  ITEM_340: printf("Voce [%s]\n",itemname[2][3]); break;
               }
               break;
       }

       /*
       **  Vediamo ora se l'utente ha effettuato una selezione multipla
       */
        id = select->NextSelect;

    }
    return(GOAHEAD);
 }

/*****************************************************************************
** H_CloseWindow: gestisce l'evento CLOSEWINDOW                             **
*****************************************************************************/
 int H_CloseWindow   (msg)  IMSG *msg; { return(CLOSEME); }
```

```
[1-HardDisk:prova] list lformat="     --> Cammino {%S} & file {%S}"
     --> Cammino {} & file {temporaneo.memo}
     --> Cammino {} & file {12Gen89.memo}
     --> Cammino {} & file {14Ago89.memo}
     --> Cammino {} & file {13Giu89.memo}
     --> Cammino {} & file {25Dic89.memo}
     --> Cammino {} & file {lettere}
[1-HardDisk:prova] cd /
[1-HardDisk:] list prova lformat="     --> Cammino {%S} & file {%S}"
     --> Cammino {prova/} & file {temporaneo.memo}
     --> Cammino {prova/} & file {12Gen89.memo}
     --> Cammino {prova/} & file {14Ago89.memo}
     --> Cammino {prova/} & file {13Giu89.memo}
     --> Cammino {prova/} & file {25Dic89.memo}
     --> Cammino {prova/} & file {lettere}
[1-HardDisk:prova] cd df0:
[1-Diskette:] list HD0:prova lformat="     --> Cammino {%S} & file {%S}"
     --> Cammino {hd0:prova/} & file {temporaneo.memo}
     --> Cammino {hd0:prova/} & file {12Gen89.memo}
     --> Cammino {hd0:prova/} & file {14Ago89.memo}
     --> Cammino {hd0:prova/} & file {13Giu89.memo}
     --> Cammino {hd0:prova/} & file {25Dic89.memo}
     --> Cammino {hd0:prova/} & file {lettere}
```

*Figura 3 - Il cammino visualizzato da «list lformat».*

```
[1] list >RAM:[temp] ????89.memo files lformat="rename %S%S %Slettere/%S.let"
[1] type RAM:[temp]
rename 12Gen89.memo lettere/12Gen89.memo.let
rename 14Ago89.memo lettere/14Ago89.memo.let
rename 13Giu89.memo lettere/13Giu89.memo.let
rename 25Dic89.memo lettere/25Dic89.memo.let
```

*Figura 4 - Come si crea un file script.*

```
/*
**  chext.rexx
**
**  CHange the EXTension by Dario de Judicibus (c) 1990 Rome - Italy
**
**  Modifica l'estensione di un file. Se ce n'è più di una, allora
**  utilizza l'ultima. Se non ce n'è alcuna, aggiungila.
**
**  Sintassi: (rx) CHEXT nome_del_file estensione
**
*/

 CSI = '9B'x                          /* Sequenza di controllo          */
 HI  = CSI||"33m"                     /* Attiva l'evidenziazione        */
 LO  = CSI||"0m"                      /* Disattiva l'evidenziazione     */

 Parse Arg file newext .
 If file = '' | ext = '' | file = '?' Then Call Help

 If  Exists(file) Then Call Msg("Non riesco a trovare" file)

 elif = Reverse(file)                 /* Inverti il nome del file.      */
 Parse Value elif With txe '.' eman   /* Separa l'ultima estensione.    */
 If eman = '' Then eman = elif        /* Non c'è? Allora si aggiunge.   */
 newfile = Reverse(eman)||'.'newext   /* Nuovo nome del file            */
 If Exists(newfile)
 Then Do
   Say newfile "esiste già. Rimpiazzo? (S/N)"
   Pull answer
   If answer  = 'S' Then Call Msg("File non rimpiazzato.")
   Address Command 'delete' newfile
 End
 Address Command 'rename' file newfile

Exit

Msg: procedure expose CSI HI LO      /* Visualizza un messaggio        */
  Parse Arg message
  Say HI||message||LO
Exit

Help: procedure expose CSI HI LO     /* Visualizza la sintassi corretta */
  Say HI||"Sintassi: (rx) CHEXT nome_del_file estensione"||LO
Exit
```

*Figura 5 - Procedura ARexx per cambiare l'estensione.*

che, basta una singola chiamata a **FreeRemember()** per permettere ad Intuition di deallocare tutti i blocchi associati ad un certo puntatore di riferimento, senza doversi preoccupare di deallocare le singole strutture. Questo ci permette di definire una sola mascherina da assegnare dopo la prima allocazione, e di evitare così di tener traccia di quelle successive. A questo punto, qualora **CloseAll()** venga chiamata, vuoi per il normale completamento del programma, vuoi nel caso si sia verificata una situazione di errore non ricoverabile (cioè quando il programma non può più andare avanti), essa non farebbe altro che verificare se *almeno una allocazione* è stata effettuata, per poi chiamare la procedura **FreeRemember()** che si prenderà carico di deallocare *tutte* le aree allocate, quali esse siano. Comodo vero? Grazie a questa tecnica aggiungere una nuova allocazione di memoria è semplicissimo. Basta aggiungere la relativa chiamata alla **AllocRemember()** specificando lo stesso riferimento, e verificare il puntatore di ritorno. Se nullo si chiama la **CloseAll()**, altrimenti si va avanti. Non è necessario modificare la procedura di chiusura ad ogni nuova aggiunta.

### *CloseAll()*

Qui abbiamo aggiunto due nuovi controlli: il primo serve a cancellare la struttura a menu relativa alla finestra aperta e ad annullare i segnalatori IDCMP relativi ai menu (vedi nota 2); la seconda serve appunto a deallocare tutta la memoria allocata.

### *CloseSafelyWindow()*

Di questa procedura parleremo in seguito. Per ora l'ho già aggiunta per ragioni didattiche, dato che abbiamo già accennato all'utilizzo di una singola porta utente condivisa da più finestre.

### *LetsGo()*

Anche questa, come **main(), non è stata modificata.**

### *BuildMenus()*

Questa invece è stata riempita (era vuota nella scorsa puntata). Essa chiama tre volte la **SetupMenu()** per definire i menu, e tre volte la **SetupItemList()** per definire le rispettive liste di menu. Guardando queste istruzioni, ci si rende immediatamente conto quanto sia facile aggiungere nuovi menu e nuove liste. Basta duplicare le linee in questione, modificare qua e là un indice, ed aggiungere le opportune definizioni nel blocco iniziale duplicando il codice già scritto. Da notare che, dato che comunque i puntatori ai menu ed alle voci sono già stati definiti all'allocazione della memoria, anche se le strutture sono ancora vuote, non ha molta importanza l'ordine delle sei chiamate in questione. Questo è tipico della tecnica a «scatola nera» [*black box*]. Le ultime quattro istruzioni servono ad associare i menu alla finestra, ad attivare i segnalatori IDCMP opportuni (salvando quelli vecchi per la **CloseAll()**), e ad assegnare la solita mascherina di controllo.

### *SetupMenu()*

Questa è la prima procedura nuova. Essa serve a definire una struttura **Menu** e ad aggiungerla alla lista dei menu da passare ad Intuition. Non credo che il codice richieda particolari spiegazioni. L'unico punto che tengo ad evidenziare è che è necessario fornire alla funzione anche il puntatore al menu precedente, definendo così l'ordine di comparsa dei menu, e che sia la posizione del menu, sia la sua larghezza, sono calcolate automaticamente.

### *SetupItemList()*

Un po' più complessa della precedente, questa funzione definisce una lista di voci, calcola automaticamente la posizione dell'elemento, la sua larghezza, anche in funzione dell'esistenza o meno di un simbolo marcatore [*checkmark*] e/o di un comando scorciatoia [*shortcut*], e lega ogni voce alla precedente nella lista.

Per il momento assume che tutte le voci siano indipendenti (**MutualExclude** nullo), che non ci sia una immagine alternata, e che l'elemento sia di tipo testo. È tuttavia già predisposta ad accettare una lista di sottovoci.

### *HandleEvent()*

Anche questa procedura è rimasta immutata, graze alla tecnica dei tronconi [*stub routine*].

### *STUBs*

Sono rimaste solo due, una per gli eventi **MOUSEBUTTONS**, ed una per quelli di tipo **MENUVERIFY**. Da notare che, mentre già possiamo ricevere i primi, i secondi sono per il momento disattivati. Grazie alle procedure vuote, tuttavia, possiamo già girare il programmare così com'è senza problemi.

### *H_MenuPick()*

Questa è l'ultima delle nuove procedure, ed ancora incompleta, sebbene perfettamente funzionante. È il cuore della gestione dei menu; è cioè la procedura che identifica la o le voci selezionate, e passa quindi il controllo al codice che deve effettuare quanto richiesto dall'utente via menu.

Quella presentata è la tecnica più completa, quella cioè che tien conto anche delle selezioni multiple, come è facile verificare lanciando il programma. Essa si riferisce al caso di una sola finestra, ma può essere opportunamente generalizzata per gestire più finestre che condividono la stessa porta utente, evitando così di aggiungere del codice ridondante per ogni finestra.

In pratica si tratta di effettuare un ciclo in cui vengono analizzati tutti i codici relativi ad una determinata selezione. Il primo viene ottenuto dal campo **Code** del messaggio arrivato. Da quest'ultimo si ricavano anche l'indirizzo della finestra relativa al menu sotto analisi, e da qui il puntatore alla prima struttura menu e quindi, di fatto, all'intero albero dei menu. Tale puntatore ci permette di ricavare il puntatore alla struttura voce relativa al codice in oggetto, grazie alla funzione **ItemAddress()**, e da questa ottenere eventualmente il codice successivo dal sottocampo **NextSelect** nel caso si tratti di selezione multipla. Avremmo potuto ovviamente utilizzare direttamente il puntatore all'albero dei menu **&menulist[0]**, ma questo avrebbe reso meno flessibile e sicura la procedura **H_MenuPick()**, dato che, nel caso di più finestre, e quindi di più menu, avremmo comunque dovuto utilizzare questa tecnica per capire a quale finestra, e quindi a quale menu, il messaggio fa riferimento.

A questo punto c'è il blocco di gestione dell'evento vero e proprio. In pratica si tratta di una scansione sugli identificativi dei menu e delle voci (se ci fossero, anche delle sottovoci), effettuata tramite strutture **switch/case**. Nel nostro caso, per il momento, a fronte di un menu e di una voce, ci limitiamo a stamparne il nome via **stdout**. Ovviamente è qui che vanno le chiamate alle singole funzioni o macro responsabili di gestire la richiesta (o le richieste) dell'utente.

### *Conclusione*

Nella prossima puntata continueremo ad analizzare le possibilità offerteci da Intuition per quanto riguarda i menu. Ovviamente continueremo ad usare ed estendere lo scheletro fin qui ottenuto in modo da avere sempre un esempio funzionante delle funzionalità che verranno di volta in volta presentate. Dato però che in tal modo il codice crescerà sempre di più, non potrò più riportarlo per intero, ma solo per differenze rispetto a quello della puntata precedente. Conservate con cura quindi questa puntata di *Programmare in C su Amiga*, perché molto probabilmente è l'ultima in cui è mostrato lo scheletro per intero. MC

# La scheda tecnica

Chi è abituato a lavorare con il CLI si trova spesso nella necessità di compiere una determinata operazione su un certo numero di file i cui nomi soddisfano lo stesso pattern, ma di non poterlo fare perché il comando in questione non supporta i caratteri speciali di sostituzione [*wildcard*]. In alcuni casi si può ricorrere ad uno dei tanti programmi di gestione del *filing system*, come **DirUtil**, o **DiskMaster**, altre volte si possono utilizzare *shell* o comandi PD che permettono di fornire ai comandi di AmigaDOS che ne sono sprovvisti la capacità di supportare i caratteri di sostituzione (come ad esempio **With** — dal disco #253 della collezione di *Fred Fish*).

Esistono tuttavia delle operazioni concettualmente semplici che mettono spesso in difficoltà anche l'utente più smaliziato. Ad esempio, giorni fa mi ha telefonato un amico dicendomi: *«Sto cercando di rinominare tutti i file di un direttorio aggiungendovi l'estensione* **.maurizio.** *Possibile che non esista un sistema semplice di farlo con AmigaDOS?»*.

Ebbene, il sistema esiste, ed è pure molto potente. Esso sfrutta l'opzione **lformat** del comando **list**. Vediamo in che modo.

Supponiamo di essere in un direttorio chiamato **prova**, e lanciamo il comando **list** senza alcun parametro:

```
[1] list
temporaneo.memo          2892 ----rwed Today    21:33:50
12Gen89.memo              948 ----rwed Today    21:33:55
14Ago89.memo             7556 ----rwed Today    21:33:47
13Giu89.memo             9704 ----rwed Today    21:33:54
25Dic89.memo              242 ----rwed Today    21:33:45
lettere                   Dir ----rwed Today    21:34:00
5 files - 1 directory - 51 blocks used
```

Supponiamo inoltre di voler cambiare tutti i file di tipo *memo* in file di tipo *lettere* e di volerli poi spostare nel sottodirettorio *lettere*, ad eccezione del promemoria temporaneo.

Si tratta di una operazione un pochino più complessa di quello che mi aveva chiesto il mio amico, ma abbastanza comune.

Il comando **rename** dell'AmigaDOS non ci viene decisamente in aiuto. Anche comandi come quelli menzionati sopra richiederebbero comunque un intervento manuale alquanto noioso, specialmente se i file, invece di essere cinque, fossero cinquanta. Il problema non è tanto che sia difficile specificare con i caratteri di sostituzione il pattern di scansione, quanto che non esiste alcun modo di dire al computer «Nel nuovo nome usa quel determinato pezzo del vecchio nome, cioè cambia tutti i file del tipo **?????89.memo** in file del tipo **=====89. lettera,** dove ===== deve corrispondere agli stessi caratteri che soddisfano **?????**».

Vediamo ora che cosa fa l'opzione **lformat** del comando **list**. Vediamo prima un esempio pratico; proviamo a scrivere:

```
[1] list lformat="Ho trovato il file %S!"
```

Il risultato è il seguente:

```
Ho trovato il file temporaneo.memo!
Ho trovato il file 12Gen89.memo!
Ho trovato il file 14Ago89.memo!
Ho trovato il file 13Giu89.memo!
Ho trovato il file 25Dic89.memo!
Ho trovato il file lettere!
```

Molto carino, ma a che serve? E poi **lettere** non è un file! Alla seconda domanda è facile rispondere: non abbiamo detto a **list** di visualizzare *solo* i file. Basta aggiungere l'opzione **files** ed il gioco è fatto. In quanto alla prima domanda, vediamo esattamente quale è il formato di questa opzione:

**LFORMAT="stringa"**

La stringa di caratteri segue le seguenti regole:

- il segno di uguale non deve essere preceduto o seguito da spazi.
- La stringa di definizione del formato da visualizzare deve *sempre* essere inclusa tra doppie virgolette.
- La stringa può contenere qualsiasi carattere tranne le *<doppie virgolette>* **["]** coppia <percento> <S> **[%S].** Le prime possono essere ottenute utilizzando il carattere speciale *<asterisco>*, e cioè **[*"]**, la seconda può essere ottenuta raddoppiando il segno del *<percento>*, **[%%S].**
- Se la stringa contiene una o più coppie **%S**, il comando opera le seguente sostituzioni:

**una coppia**
viene sostituita dal nome del file trovato;
**due coppie**
la prima viene sostituita dal cammino per raggiungere il file a partire dal direttorio corrente, la seconda dal nome del file trovato;
**tre coppie**
la prima viene sostituita dal cammino per raggiungere il file a partire dal direttorio corrente, la seconda; e la terza dal nome del file trovato (vedi nota 3);
**quattro coppie**
la prima e la terza vengono sostituite dal cammino per raggiungere il file a partire dal direttorio corrente, la seconda e la quarta dal nome del file trovato;
**cinque o più coppie**
il comportamento è lo stesso che nel caso di quattro coppie; le altre vengono ignorate e sostituite da una stringa nulla.

- Il cammino termina sempre con due punti, se limitato ad un volume o ad un device, con la barra diagonale [*slash*] se contiene una lista di direttorii (ad esempio **df1:**, **hd0:prova/lettere/**).

Notare che quando si parla di cammino per raggiungere il file, si intende quello a partire dal direttorio dal quale si è lanciato il comando. Se quindi stiamo listando il direttorio corrente, questo è di fatto nullo, come si può vedere negli esempi di figura 3.

Vediamo allora come risolvere il nostro problema. Vi ricordo che il nostro scopo è quello cambiare tutti i file di tipo *memo* in file di tipo *lettere* e di spostarli poi nel sottodirettorio **lettere**, ad eccezione del promemoria temporaneo, che non va né rinominato, né spostato.

Il trucco è il seguente (seguire su figura 4). Innanzi tutto si definisce il pattern di scansione, e cioè **?????89.memo**, che seleziona tutti e solo i file che ci interessano. Poi si esegue il comando **list** come in figura, avendo l'accortezza di usare l'opzione **files**. Il risultato va reindirizzato sulla **RAM**: e da lì eseguito come un file di tipo *script*.

In realtà non sempre questo basta. Il problema è che possiamo prendere dal vecchio file solamente il nome intero, non una parte del nome. Nel nostro caso i nuovi file nel direttorio **lettere** vengono ad avere la doppia estensione **.memo.let**, mentre magari noi volevamo solo la seconda. Per far questo è ancora necessario operare manualmente entrando nel file **RAM:[temp]** (vedi nota 4) con un editore di testi e cambiare tutte le occorrenze **.memo.let** in **.let**. Esistono anche dei programmini che permettono di far questo da CLI, senza entrare in edizione del file, alcuni sono PD, altri, come **SPLAT** sono inclusi in altri prodotti (Lattice C). Quest'ultimo comunque sembra avere ancora dei bachi seri, anche se in un modo o nell'altro si riesce ad utilizzarlo.

Certo, questo secondo passaggio è scocciante, sarebbe stato meglio poter fare tutto in una botta sola. E allora? Allora l'unica è usare l'ARexx. In figura 5 è riportata una procedura ARexx che ho scritto qualche giorno fa e che serva a modificare l'ultima estensione di un file od ad aggiungervela, se questo non esiste. A questo punto basta utilizzare la sequenza di comandi:

```
[1] list >RAM:[temp] ?????89.memo files lformat="rx chren %S%S let"
[1] list >>RAM:[temp] ?????89.let files lformat="rename %S%S %Slettere/%S"
[1] execute RAM:[temp]
```

Naturalmente se nel direttorio corrente c'erano già dei file di tipo lettera che non volevamo spostare, il risultato non è proprio quello voluto, mam l'importante è capire il meccanismo, dopodiché ognuno lo potrà adattare alle esigenze del momento. Tra l'altro, se il numero di file è elevato, vale sempre la pena di scrivere una procedura ARexx per risolvere il problema, per quanto specifico sia. Come? Non conoscete l'ARexx? Male, molto male... Chissà, forse ne riparleremo in futuro. Per quelli invece che l'ARexx lo conoscono già, un esercizio. Come potrete facilmente verificare, la macro riportata in figura non gestisce propriamente i file il cui nome contiene spazi o caratteri speciali, quelli cioè che in AmigaDOS si devono specificare tra virgolette.

Provate a modificare **chext.rexx** per supportare anche questo tipo di file.